AMENDMENTS TO THE SPECIFICATION

[0026] Referring now to Figure 3, an architectural system diagram depicting the operation of a JAVA virtual machine within a conventional data processing system such as computer system 200 of Figure 2a is illustrated. A system 300 may include an operating system 314 allocating access to hardware 316 resources and may also include a JAVA ~~Java~~ Virtual Machine (JVM) implementation 304 capable of executing JAVA ~~Java~~ programs in bytecode 302 format as depicted in Figure 3. The illustrated JAVA ~~Java~~ Virtual Machine 304, running on a client or server computer system 300, relies on services and hardware resources 316 such as registers, JAVA stacks, a heap, and a method area provided by underlying operating system 314 to execute JAVA programs. The JAVA ~~Java~~ Virtual Machine 304 may utilize a JAVA ~~Java~~ Interpreter 306 or a JAVA ~~Java~~ "Just-In-Time" (JIT) compiler 308 to generate executable native code 310 from a received JAVA ~~Java~~ bytecode class file 302.

[0027] In a networked environment, a user first accesses a server such as server 100 of Figure 1 through a network such as network 110 and retrieves or downloads a desired JAVA ~~Java~~ class file 302 into a client computer system 300 such as one of clients 150 of Figure 1. After the class file 302 has been downloaded, the JAVA ~~Java~~ Virtual Machine 304 verifies the class file to ensure that the program will not cause security violations or cause harm to computer system resources. After the JAVA ~~Java~~ program has been verified, a JAVA ~~Java~~ JIT compiler 308 may compile the JAVA ~~Java~~ class file 302 and generate executable native processor code 310. Then this dynamically compiled code 310 may be executed directly on computer hardware 316. In order to maintain the state of the JAVA ~~Java~~ Virtual Machine 304 and make system calls during

the above-described process, native code 310 and JAVA Virtual Machine 304 communicate using various calls 312.

[0030] Referring now to Figure 5, a block diagram of a runtime environment data area 500 of a conventional JAVA virtual machine is illustrated. The JAVA system includes support for the simultaneous operation of multiple program contexts or "threads" and as each thread begins, a JAVA stack 502-506 is created and associated with it by the JAVA Virtual Machine. Each JAVA stack 502-506 is composed of one or more stack frames each containing the state of one JAVA ~~Java~~ method invocation. The state of a JAVA ~~Java~~ method invocation includes its local variables, the parameters with which it was invoked, its return value (if any), as well as intermediate calculations. Figure 5 depicts a JVM runtime environment in which three threads are executing, each with its own associated JAVA stack 502-506. It will be appreciated that greater or fewer threads may exist in any given JVM implementation or instance. The method that is currently being executed by each thread is that thread's current method and the stack frame for the current method is the current frame 508. When a thread invokes a JAVA ~~Java~~ method, the virtual machine creates and pushes a new frame onto the thread's JAVA ~~Java~~ stack. This new frame then becomes the current frame 508. When a method completes, whether normally or abruptly, the JAVA ~~Java~~ virtual machine pops and discards the method's stack frame and the frame for the previous method becomes the current frame 508. As shown in the illustrated embodiment, a stack frame 510 may be used to store parameters and local variables 512, operands 514, intermediate computations, and other data such as data identifying the frame's associated method, the invoking method's frame, stack pointers, and program counters. The JAVA ~~Java~~ Virtual Machine has no registers to hold intermediate data values but rather uses the above-

described stacks to store intermediate data values. This approach was taken by the designers of the JAVA system to keep the virtual machine's instruction set compact and to facilitate implementation on architectures with few or irregular general purpose registers.